CS29003 ALGORITHMS LABORATORY
(Divdede and Conquer – Solution)
Date: Sep 12 2020

1 Unimodal Search

The approach will be similar to Binary Search

Notice that by the definition of unimodal arrays, for each $1 \leqslant i < n$ either $A[i] < A[i + 1]$ or $A[i] > A[i + 1]$. The main idea is to distinguish these two cases:

1. By the definition of unimodal arrays, if $A[i] < A[i + 1]$, then the maximum element of $A[1..n]$ occurs in $A[i + 1..n]$.

2. In a similar way, if $A[i] > A[i + 1]$, then the maximum element of $A[1..n]$ occurs in $A[1..i]$.

This leads to the following divide and conquer solution (note its resemblance to binary search):

Algorithm 1 Algorithm for problem 1

```
 1:  a, b = 1, n
 2:  while a<b do
 3:      mid = ⌊(a + b)/2⌋
 4:      if A[mid] < A[mid + 1] then
 5:          a = mid + 1
 6:      end if
 7:      if A[mid] > A[mid + 1] then
 8:          b = mid
 9:      end if
10:      return A[a]
11:  end while
```

The divide and conquer approach leads to a running time of $T(n) = T(n/2)+\theta(1) = \theta(\lg n)$.

2 Convex Polygon

Notice that the x-coordinates of the vertices form a unimodal array and we can use part 1 to find the vertex with the maximum x-coordinate in $\theta(\lg n)$ time.

After finding the vertex V [max] with the maximum x-coordinate, notice that the y-coordinates in V [max], V [max + 1], . . ., V [n - 1], V [n], V [1] form a unimodal array and the maximum y-coordinate of V [1..n] lies in this array. Again part 1 can be used to find the vertex with the maximum y-coordinate. The total running time is $\theta(\lg n)$.

3 Median Of Sorted Arrays

This method works by first getting medians of the two sorted arrays and then comparing them.

Time Complexity: $\mathcal{O}(\log n)$

Algorithm 2 Algorithm for Problem 3
___
1: Calculate the medians m1 and m2 of the input arrays arr1[] and ar2[] respectively
2: **if** m1 == m2 **then**
3:     return m1 (or m2)
4: **end if**
5: **if** m1 > m2 **then**
6:     From first element of arr1 to m1 (arr1[0..n/2])
7:     From m2 to last element of ar2 (arr2[n/2..n-1])
8: **end if**
9: **if** m2 > m1 **then**
10:     From m1 to last element of ar1 (arr1[n/2..n-1])
11:     From first element of ar2 to m2 (arr2[0..n/2])
12: **end if**
13: Repeat the above process until size of both the subarrays becomes 2
14: **if** size of the two arrays is 2 **then**
15:     Median = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2
16: **end if**
___

4   A bag of rice

We can see that bag will be full for starting $(l + 1)$ days because rice taken out is less than rice being filled. After that, each day rice in the bag will be decreased by 1 more kilogram and on $(l + 1 + i)$th day $(C – (i)(i + 1) / 2)$ kilogram rice will remain before eating rice. Now we need to find a minimal day $(l + 1 + K)$, in which even after filling the bag by l kilograms we have rice less than l in bag i.e. on $(l + 1 + K – 1)$th day bag becomes empty so our goal is to find minimum K such that, $C – K(K + 1) / 2$ i= l

We can solve above equation using binary search and then $(l + K)$ will be our answer. Total time complexity of solution will be $\mathcal{O}(\log C)$

Algorithm 3 Algorithm for Problem 4
___
1: Func getCumulateSum(n) ← return (n*(n+1)/2)
2: **if** C⩽l **then**
3:     return C
4: **end if**
5: int lo = 0; int hi = C; int mid;
6: **while** lo < hi **do**
7:     mid = (lo + hi) / 2;
8:     **if** getCumulateSum(mid) ⩾ (C - l) **then**
9:         hi = mid;
10:     **else**
11:         lo = mid + 1;
12:     **end if**
13: **end while**
14: return (l + lo);
___